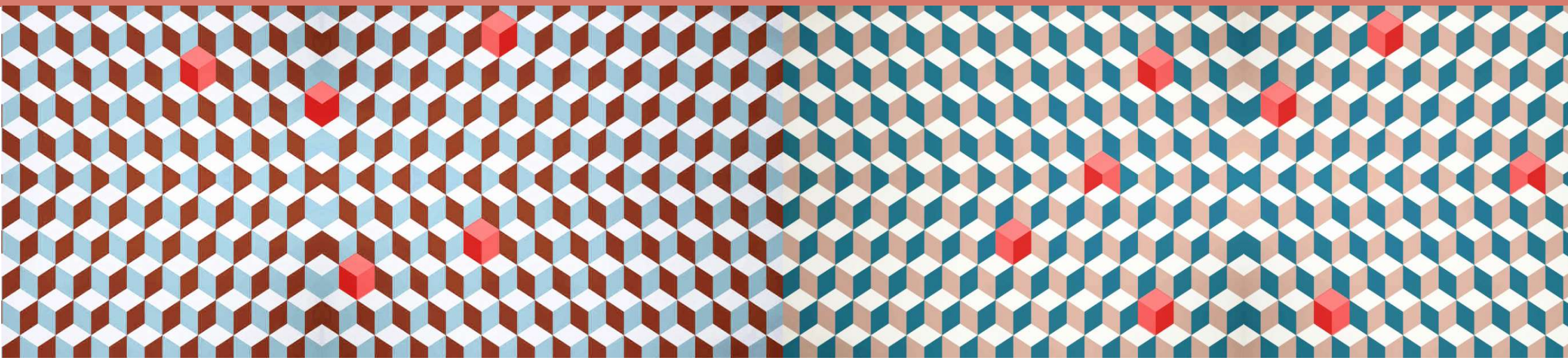
Sincronizzazione:

Esercitazione e concetti

Tutor: Giovanni Hauber



Esercizio

- In un ristorante self-service, i clienti, dopo aver mangiato, dispongono i vassoi in M contenitori, ognuno di K ripiani.
- **Periodicamente**, un addetto sceglie un contenitore **tra quelli in cui ci sono più ripiani liberi**, lo svuota, lava i piatti e riporta il contenitore in sala.

Consiglio: I contenitori in cui riporre i vassoi sono M ed ognuno di essi ha K ripiani (i.e: 5 contenitori, ognuno dei quali ha 3 ripiani liberi). Questo può essere risolto usando

- un'array 2D contenitori[M][K] e l'accesso in mutua esclusione ad esso per il controllo
- un'array di semafori contatori con una struttura ausiliaria per il controllo (a cui si accede sempre in mutua esclusione).

Possibile soluzione: Main

```
main() {

  crea_processo(addetto)

  // Nota: J > M!
  for i=0 to J:
    crea_processo(Cliente)

}
```

```
// Semafori contatore: il primo rappresenta i contenitori
disponibili, il secondo invece i ripiani per lo specifico
contenitore (array di semafori contatore)
semaforo contatore: contenitore_disponibile=M;
semaforo contatore: ripiano_disponible[M]=K;

// Mutex per l'accesso alla sezione critica
mutex:  accesso_risorse=1;

// Indice che tiene conto del contenitore a cui stiamo facendo
accesso
int primo_contenitore_libero=0;

// Struttura ausiliaria che tiene conto dell'M-simo
contenitore quanto risulti pieno, per facilitare il controllo
da parte dell'addetto
int ripiani_contenitori[M]=0

// Costante
int VASSOIO=1
```

Possibile soluzione: Cliente

```
semaforo contatore: contenitore_disponibile=M;
semaforo contatore: ripiano_disponibile[M]=K;
mutex:  accesso_risorse=1;
int primo_contenitore_libero=0;
int ripiani_contenitori[M]=0
int VASSOIO=1
```

```
cliente(){
  while(1) {
    wait(contenitore_disponibile);

   // Prendo la mutua esclusione, faccio una copia locale ed incremento
    lock(accesso_risorse)
    contenitore_loc = primo_contenitore_libero ++
    unlock(accesso_risorse)

    // La wait simula la presa di quello specifico ripiano, ed accediamo
    // all'array con un indice unico (quindi non ce necessita di mutex)!
    wait(ripiano_disponibile[contenitore_loc])
    ripiani_contenitori[contenitore_loc] += VASSOIO;

    // Non c'è la signal sul ripiano poiche è l'addetto che lo svuota!

    lock(accesso_risorse)
    primo_contenitore_libero -= 1 // decremento l'indice del contenitore incrementato prima
    unlock(accesso_risorse)

    signal(contenitore_disponibile)
  }
}
```

Possibile soluzione: Addetto

```
semaforo contatore: contenitore_disponibile=M;
semaforo contatore: ripiano_disponibile[M]=K;
mutex:  accesso_risorse=1;
int primo_contenitore_libero =0;
int ripiani_contenitori[M]=0
int VASSOIO=1
```

```
addetto() {
Repeat
  // «periodicamente svuoto»
  sleep(rand(0.00001, 0.1))
  min = inf
  ind = -1
  // Per ogni ripiano (M IN TOTALE)
  for(i=0;i<M;i++){
    lock(mutex) // nota: qui il mutex potrebbe essere opzionale!
    if (ripiani_contenitori[i] <= min && ripiani_contenitori[i] != 0) {
    unlock(mutex)
    ind = i
    min = ripiani_contenitori[i]
   } else unlock(mutex)
  }

   if (min != inf) {
    for(i=0;i<min;i++) signal(ripiano_pieno[i])
  }

Forever
}
```

Esercizio

- In una pizzeria, **due pizzaioli** sfornano continuamente pizze che pongono di volta in volta in uno di **N piatti disponibili** su un **bancone**. I piatti sono **prelevati da M (M<N) camerieri** che provvedono a servirle ai tavoli. Ciascun cameriere può portare:
- *2 pizze per volta, se disponibili*;
- *ma, in caso di affollamento, i camerieri preleveranno 3 pizze per volta*.

Fornire una soluzione con semafori e discutere eventuali problemi di starvation/deadlock.

Possibile Soluzione: Main

```
Bancone: Array[N] = {0}
Posizione_Bancone: Integer = 0
Clientela: Integer = 0
Affollato: Boolean = False
Pieno: Semaforo Contatore = 0
Vuoto: Semaforo Contatore = N
CS: Mutex = 1
CS_Clientela: Mutex = 1
Aspetta_Pizza: Semaforo Binario = 0
```

```
Main() {
crea_processo(Clientela_Manager)
for i = 0 to 2:
        crea_processo(Pizzaioli)
for i = 0 to M:
        crea_processo(Camerieri)
}
```

Possibile Soluzione: Clientela

```
Bancone: Array[N] = {0}
Posizione_Bancone: Integer = 0
Clientela: Integer = 0
Affollato: Boolean = False
Pieno: Semaforo Contatore = 0
Vuoto: Semaforo Contatore = N
CS: Mutex = 1
CS_Clientela: Mutex = 1
Aspetta_Pizza: Semaforo Binario = 0
```

```
Clientela_Manager() {
 Repeat
      # Random sleep e aggiungiamo una persona a clientela
      sleep(rand(0.0001, 0.1))
      clientela += 1

      # Prendo il mutex della clientela e imposto affollato
      # in base al numero
      lock(CS_Clientela)
      if (clientela >= 10)
            affollato = True
      else:
            affollato = False
      release(CS_Clientela)
 Forever
}
```

Possibile Soluzione: Produttori

```
Bancone: Array[N] = {0}
Posizione_Bancone: Integer = 0
Clientela: Integer = 0
Affollato: Boolean = False
Pieno: Semaforo Contatore = 0
Vuoto: Semaforo Contatore = N
CS: Mutex = 1
CS_Clientela: Mutex = 1
Aspetta_Pizza: Semaforo Binario = 0
```

```
Pizzaioli() {
 Repeat

      # Produco una pizza
      Pizza = produciPizza()

      # Aspetto che ci siano posti sul bancone
      wait(Vuoto)

      # Accedo in sezione critica ed inserisco nell'array
      lock(CS)
      Bancone[Posizione_Bancone++] = Pizza
      release(CS)

      # Segnalo l'aggiunta di una pizza
      signal(Pieno)

 Forever
}
```

Possibile Soluzione: Consumatori

```
Camerieri() {

Pizze_da_portare[3] = {0}
Repeat
        # Prendo il mutex della clientela
        lock(CS_Clientela)
        # Se affollato, prendo 3 pizze per volta
        # altrimenti 2
        pizze_da_servire = -1
        if(affollato)
                pizze_da_servire = 3
        else
                pizze_da_servire = 2
        # Rilascio il mutex di clientela
        release(CS_Clientela)
        # Prendo il mutex del prod/cons
        lock(CS)
        …
        # Se ci sono abbastanza pizze (controllo con il numero di pizze da servire)
                if(Posizione_Bancone >= pizze_da_servire) {
                # Consumo le tre pizze, poiche disponibili
                  for i = 0 to pizze_da_servire:
                      wait(Pieno)
                      signal(Vuoto)
                      Pizze_da_portare[i] = Bancone[Posizione_Bancone--]
                  # Rilascio mutex prod cons (ho finito di decrementare)
                  release(CS)
                } else {
                # Altrimenti, consumo tutte le pizze disponibili-ci sono meno di pizze_da_servire
                  for i = 0 to Posizione_Bancone:
                      wait(Pieno)
                      signal(Vuoto)
                      Pizze_da_portare[i] = Bancone[Posizione_Bancone--]
                  # Rilascio mutex prod cons (ho finito di decrementare)
                  release(CS)
                }
                # Controllo la lunghezza di quante pizze ho preso
                lock(CS_Clientela)
                clienti_serviti = len(Pizze_da_portare != 0)
                # Decremento il numero di clienti in base a quante pizze ho
                clientela -= clienti_serviti
                # Rilascio e torno a prendere pizze
                release(CS_Clientela)
                Forever
}
```

Esercizio

Un sistema è composto di un buffer di N posizioni, 2*N processi attivi ed un processo coordinatore. Il processo coordinatore estrae uno dopo l'altro in maniera casuale i numeri da 1 a N e ad ogni estrazione i processi competono per aggiudicarsi l'accesso alla posizione corrispondente del buffer, scrivendone il proprio PID. Un processo cha ha scritto il proprio PID nel buffer non partecipa più alla contesa. Quando tutte le posizioni del buffer sono state assegnate, il coordinatore si alterna con l’ultimo processo che ha avuto accesso al buffer nel leggere e stampare, uno alla volta, il contenuto del buffer.